An Approach for Power Estimation at Electronic System Level using Distributed Simulation

Helder F. A. Oliveira[1,2,*], Alisson V. Brito[3,*], Joseana M. F. R. Araujo[1] and Elmar U. K. Melcher[1]

[1]Universidade Federal de Campina Grande (UFCG), Brazil
[2]Universidade Federal Rural do Semi-Árido (UFERSA), Brazil
[3]Universidade Federal da Paraiba (UFPB), Brazil
[*]Corresponding author: helder.oliveira@ufersa.edu.br, alisson@ci.ufpb.br

ABSTRACT

The present research aims to develop an approach using HLA (High Level Architecture), enabling the cre-ation of a distributed and heterogeneous environment, composed by different tools and models to obtain a better trade-off between accuracy and run time in power estimation. These models can be described in different languages and/or abstraction levels, as well as use different power estimation approaches. The use of HLA enables the synchronized and distributed simulation of the elements that compose the simulation environment. The approach must allow the collecting and grouping of power estimation data in a centralized manner. As a case study, an MPSoC (MultiProcessor System-on-Chip) ESL/TLM model, described in C++/SystemC, and an ESL model, created on Ptolemy framework, have been used. The experimental results show the flexibility of the approach, which promotes an integrated view of power estimation data.

Index Terms: power estimation, electronic system level, distributed simulation

I. INTRODUCTION

There are several reasons to reduce the energy consumed by an SoC: reduction of thermal wearing; cooling cost reduction of the electronic device; increase of the operating time per battery charge, among others [1].

The impact of the power generation to meet the needs of electronic devices is directly linked to the environment. This has as main consequence the emission of greenhouse gases in the atmosphere (carbon footprint) and can accelerate the process of global warming and air pollution.

Therefore, power consumption is one of the main concerns during the design of an SoC (System -on-a-Chip). Recent re-searches are dedicated to power consumption estimation in SoC design [2] [3] [4] [5] [6] [7] [8]). Each of these works is focused on a different abstraction level and is based on a different estimation approach [9].

Among the possible abstraction levels, the most important ones are the ESL (Electronic System Level), RTL (Register Trans-fer Level), Gate level and Transistor level [10]. Designers must consider power consumption issues as early as possible to re-duce development time [5]. The higher is the level of abstraction where designers can obtain information about power consump-tion, cheaper are eventual modifications in design.

In addition, high-level specifications allow the designer to ignore several RTL coding issues, such as synchronization and scheduling operations, while capturing the project features [2]. In this context, power consumption estimation at the system level (ESL) has become an important research topic in recent years. Many of the methodologies focus on the relationship between power estimation accuracy and its computational cost at ESL level [2] [5] [6] [11].

Diverse authors agree that the main challenge to develop power estimation tools at ESL is to reach the most adequate balance between simulation performance and accuracy [5] [12] [8] [9].

Due to complexity of the projects and the time pressure be-tween product specification and time-to-market, it is not always possible to realize power consumption estimation at RTL level (Register Transfer Level) or at logic gates level for a whole sys-tem. Thus, it is crucial to estimate the power consumption as soon as possible during the project of a digital circuit design.

During the last years researches proposed power consump-tion estimation approaches at ESL [2] [10]

[4] [5] [13] [9]. De-pending on the project complexity, it is necessary to combine different elements to provide an adequate power estimation, such as, different estimation approaches, different tools or, dif-ferent programming languages and different abstraction levels.

However, integrate these heterogeneous elements in a unique design environment, while exchanging data among them and keeping the overall system consistent, is a challenge and is the main focus of this work.

For this integration, High-Level Architecture – HLA offers mechanisms to enable applications, tools and simulation to ex-change information in a synchronous way [14]. HLA is defined as a common architecture for modeling and distributed simula-tion.

This research has the objective of developing an approach using High-Level Architecture (HLA) to allow the creation of a distributed and heterogeneous simulation environments, com-posed of different tools and models. These models can be de-scribed in different languages and/or levels of abstraction, and also enables different approaches to estimate the power con-sumption. The use of HLA allows simulations in a synchro-nized and distributed manner. The presented approach provides the estimation and collection of power consumption data in a centralized way. For study case, an experiment is presented where an ESL model running in Ptolemy is co-simulated with an SystemC-TLM model. Each one estimates power using a different approach and all the data is collected in centralized way.

II. RELATED WORK

The total energy consumption of an integrated circuit is the sum of dynamic and static energy consumption [15]. Techniques to reduce dynamic power became common. With CMOS process technology lower than 90nm, the static power (leakage power) has become important, and in many cases become a dominant restriction to design [16].

Techniques for reducing energy consumption can be applied at various levels of abstraction during the development of an SoC, from system specification to layout generation [15]. A sum-mary of the main techniques for energy consumption reduction can be found in [17].

As well as techniques for energy consumption reduction, power consumption estimation in an SoC can be performed at various levels of abstraction. According to [18], the power consumed by a digital circuit can be estimated at four differ-ent levels of abstraction: transistor-level, gate-level, Register Transfer Level (RTL) and Electronic System level (ESL).

The main challenge for power consumption estimation tools is to achieve the most adequate balance between performance and accuracy [5] [8] [9]. To reduce energy consumption in a final product of a hardware design, it is beneficial that the power consumption estimation can be performed since the very beginning of the design flow [9].

Tools for power consumption estimating at lower levels of abstraction allow accurate energy modeling, but result in longer simulation run time and higher design cost [5] [1].

Methods for power estimation with a low level of abstrac-tion (e.g., layout, gate level and RTL) take into account many details of the simulated SoC, making simulations slower and in-creasing the design time. The slowness of these methods can be considered as an obstacle to productivity [9]. Thus, estimation techniques at a higher level of abstraction (e.g., ESL) are desired.

ESL power consumption estimation techniques perform esti-mations based on system descriptions at higher level of abstrac-tion using energy consumption models (power models).

A power model can be defined as a model that captures the dependence of power dissipation of a design block based on certain parameters, such as switching activity, capacitance, etc. Its accuracy is very dependent on the model of computation, input/output activities, capacitance, etc [2].

Many works in power estimation scope try to focus on esti-mating the power consumption using different power models, in order to provide more accurate power estimation. These power models usually vary in terms of granularity [19] [13] [20] [21], allowing the designer to decide which use to achieve a good trade-off between accuracy and efficiency.

In [19] the authors propose a power estimation framework that integrates heterogeneous component power models us-ing a network of "power monitors". Power monitors enable each component model to be associated with multiple (distinct) power models of differing accuracy and efficiency, or with con-figurable power models which can be tuned to different accu-racy/efficiency levels.

Authors from [13] have developed a system-level power esti-mation framework which uses different power modeling tech-niques for each component: processor cores, bus fabrics, custom IP blocks and memories. The authors highlight the fact that an SoC has many heterogeneous components with varying power characteristics, ranging from very regular structures such as on-chip SRAM to irregular custom IP blocks such as video codec, what makes difficult to derive a single modeling methodology that covers every component constituting an SoC device. Thus, different approaches need to be adopted for different compo-nents.

In another work [20] is presented a hierarchical power charac-terization model that is applicable to any kind of custom IP. The power model has several levels that have varying accuracy, sim-ulation speed and design effort. These levels define the power models accuracy. Upper levels represent coarser grained power models, while lower levels represent more accurate power mod-els, i.e., with greater detail. The methodology was applied on an H.264 decoder prediction IP.

Similarly, in [21] was developed an ESL power estimation framework based on an interface which allows to integrate vari-ous power models. Designers can choose either the fine-grained or coarse-grained power models according to the trade-off be-tween accuracy and computing cost. For evaluating power esti -mation framework, PAC (Parallel Architecture Core) Duo system [22] was adopted as the study case.

Several works concerning power estimation at ESL (Elec-tronic System Level) or RTL (Register Transfer Level) implement their approach extending the SystemC functionality, by modi-fying the SystemC library [23] or adding new libraries [24] [25] [26].

The authors in [23] modify the SystemC class library to allow the calculation of energy consumption of hardware described at ESL. According to the authors, the SystemC class library was modified in order to receive a new set of classes for power esti-mation, so called Powersim. The Powersim operation is based on monitoring the C++ operators, when called on SystemC data type. Different energy models are used for each data type. This method does not require any change in the application source code to obtain the energy consumption during a SystemC simu-lation.

In [24] the SystemC class library was extended with new classes describing area, delay and power characteristics of the SystemC RTL models.

A framework called PowerSC is presented in [25]. Here, the authors instruments SystemC for power characterization, mod-eling and estimation. The key idea was to extend SystemC by adding power -aware C++ classes. The PowerSC framework pro-vides an API to support the integration of alternatives estimation techniques. To use the framework, only two modifications are necessary in the SystemC description to enable PowerSC usage: The inclusion of the main PowerSC header file within the model files, and the invocation of a PowerSC macro in the end of the simulator's main function, in order to print out the results. The PowerSC framework can be used at ESL, RTL or gate-level.

In [26] the authors developed a library which extends every SystemC module with non-functional data regarding physical layout and power consumption and which accumulates and estimates dynamic energy usage. The work is based in two pre-vious libraries for SystemC power modeling: TLM POWER2 [27] and PKtool [28]. The work supports both phase/mode power modeling and energy-per-transaction logging for TLM (Transactional-Level Modelling). The state of an IP block is char-acterized by both its phase and mode. A phase is characterized by its power and time duration (e.g. read, wait, compute), and mode, the actual operation mode (e.g. on, off, sleep).

Depending on the project requirements, to guarantee a better accuracy in power estimation, it might be necessary to estimate the power consumption of a system or part of it using differ-ent elements: different power estimation approaches, tools or, even, models described in different languages and/or abstrac-tion levels. Nevertheless, it is a challenge to incorporate these elements to create a simulation environment distributed and heterogeneous, which allows these elements to communicate and exchange information synchronously.

The SoC designer needs to evaluate which abstraction levels and estimation techniques are the most adequate for each part of the SoC in the particular application context. Then a frame-work is needed, which enables the designer to integrate these abstraction models and power estimation techniques into one unique simulation that provides centralized information on the overall power consumption.

To the best of our knowledge, none of the aforementioned works uses a general purpose architecture for modeling and dis-tributed simulation, such High -Level Architecture (HLA), in or-der to allow the creation of a heterogeneous and distributed sim-ulation environment, composed by different tools, models de -scribed in different languages and/or abstraction levels, which enables the use of different power estimation approaches for each model during the simulation.

A. High-Level Architecture (HLA)

In order to allow a simulation environment distributed and het-erogeneous, composed by different tools, models described in different languages and/or abstraction levels, adopting different power estimation techniques, we used the High-Level Architec-ture (HLA) in our approach.

High-Level Architecture (HLA) is a general -purpose archi-tecture defined by Defence Modelling and Simulation Office (DMSO) to support reusability and interoperability using a wide number of different types of simulators, which are maintained by U.S Department of Defense (DoD) [14]. The standard defines this architecture having three parts: the first explores the frame-work in a general way and its main rules, the second deals with the interface specifications between simulators (HLA), and the third deals with the specification model of data (OMT) which are transferred between simulators.

The HLA has a time managing service to synchronize data between heterogeneous models. The main goals of HLA are to enable the interoperation of distinct models and reuse them when necessary, to provide a distributed simulation environ-ment for systems that need large scale computing.

The principal idea of HLA is to separate the specifics function-ality of each simulator using a general proposal infrastructure (see Figure 1). Each simulator needs to use the RunTime Infras-tructure (RTI) to communicate to HLA and others simulators. The RTI is responsible for the specific structures of each simula-tor to interface with the global structure of HLA. Each simulator that is connected to one RTI is called a Federate. The set of all federates managed by one RTI is called a Federation. In cases of geographically distributed simulations, it is also possible to have many Federations in a same simulation environment.

Analogous to how a distributed operating system provides services to applications, the RTI provides services to federates. These services are used, basically, for synchronizing the feder-ates and controlling the data exchange between them. These interfaces are arranged into seven basic service groups: Federa-tion management, Declaration management (DM), Object man-agement, Ownership management, Time management, Data distribution management (DMM) and Support services (refer to [29] for details).

In our approach, each Federate can be an independent sim-ulator (i.e., SystemC, Matlab, Ptolemy, etc.) or even part of a system running in a simulator.

In HLA, an object-oriented paradigm is used to describe data, called Federate Object Model (FOM). There, it is possible to describe classes, objects, attributes and hierarchy of classes. Once configured, all federates read and write attributes of the objects defined by FOM. HLA uses publish-subscribe protocol, thus every federate must announce the subscribed object and which object it can publish. In our approach, all the federates subscribe and publish all objects and attributes. The description of these objects is presented following. For more details about Federation Object Model (FOM) rules and syntax, refer to [29].

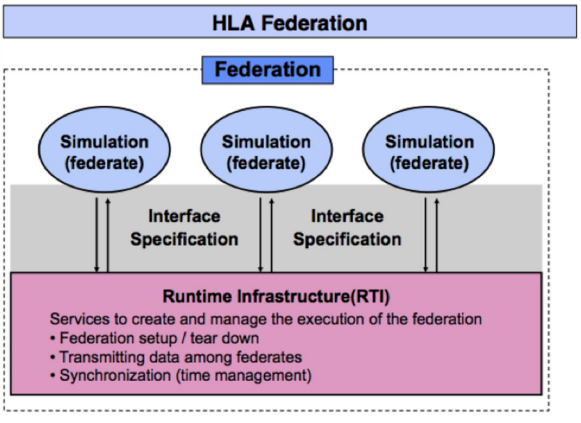

Figure 1. Architecture of HLA Federation

```
(FED
  (Federation TestFed)
  (FEDversion v1.3)
  (spaces)
  (objects
    (class PowerHLA
      (attribute privilegeToDelete reliable timestamp)
      (class RTIprivate)
      (class module
        (attribute id reliable timestamp)
        (attribute energy reliable timestamp)
        (attribute module reliable timestamp)
        )))
  (interactions))
```

B. MPSoC Modeling

In our case study we use a benchmark composed of a scalable set of MPSoCs described in C++/SystemC, called MPSoCBench [30]. MPSoCBench is a simulation toolset composed of a scal-able set of MPSoCs useful for the development and high level evaluation of new tools, methodologies, software, and hardware components. This tool provides a complete open source simu-lation infrastructure including scalable hardware and software components, with easy instrumentation and fast simulation at different abstraction levels.

The toolset supports four different ISAs in many configurable and scalable MPSoC platforms with up to 64 cores, with differ-ent interconnections at different simulation abstraction levels. The user provides parameters to create MPSoC simulators and the script generates appropriate code for processors, intercon-nections, caches, memories, and IPs. These components are compiled to create one or more MPSoC simulators that will be stored in specific folders. The set of applications (e.g., Secure Hash Algorithm, Dijkstra, Fast Fourier Transform, etc.) are com-piled with the appropriate cross-compilers, and the executable files are stored together with the simulators. After executing each simulation, a proof of correctness is performed evaluating the application output files against a golden result.

The MPSoCBench includes SPARC and MIPS processors mod-els based on the LEON3 and PLASMA RTL models. The power consumption model for these processors use PowerSC [25] framework. This framework instruments SystemC for power characterization, modeling and estimation in multiple abstrac-tion levels. Its basic idea is to extend the SystemC library, adding C++ classes which enable power estimation.

The power models are based on different FPGA and ASIC technologies. Some key usages are power

consumption evalu-ation during program execution, detecting power bottlenecks, and comparison among different architectures.

Using the PowerSC framework, when a SystemC class is compiled, an augmented executable specification is generated (by linking both, the SystemC and PowerSC libraries), instead of a conventional executable specification (by linking the SystemC library only). The augmented specification is instrumented to gather signal statistics during simulation.

When the simulation is launched, the design elements are monitored and power information is dynamically recorded. At the simulation end, the resulting information is summarized in power reports.

The authors of PowerSC highlight that the effort to enable the power estimation mechanism is minimal, since SystemC data types, signals and modules do not need to be manually changed, because the PowerSC framework modifies them automatically.

In our case study, an MPSoC SystemC TLM model built by the MPSoCBench communicates with an ESL Ptolemy model via HLA.

C. Embedded System Modeling

Due to its close coupling with the environment, Embedded Sys-tems normally include different components with heterogeneous Models of Computation (MoC). Thus, tools with high abstrac-tion power are necessary in order to model, simulate and test all such MoCs, e.g., Finite State Machines (FSM), Synchronous Data Flow (SDF), Discrete Events (DE) and Continuous Time (CT). The Ptolemy II framework [31] is an example of a simulation and modeling tool intended to support system designs that involve combining different MoCs.

In Figure 2 is possible to see the main components presented by Ptolemy's GUI, called Vergil. The Director manages the simu-lation according to its Model of Computation. It is also possible to see that there are two kinds of actors. The Atomicare the indivisible ones and the Composite Actors can aggregate other actors or complete designs (also with different Models of Com-putation) inside it. These hierarchical actors are the strategy of Ptolemy to have heterogeneous Models of Computation in a same simulation model. The actors exchange data among them (called Tokens) via Ports and Relations.

In this work, Ptolemy is used as one of the simulators in the proposed approach and the power consumption is estimated by random values produced only as proof of concept. Ptolemy was integrated to HLA using a prior development by our research group [32].

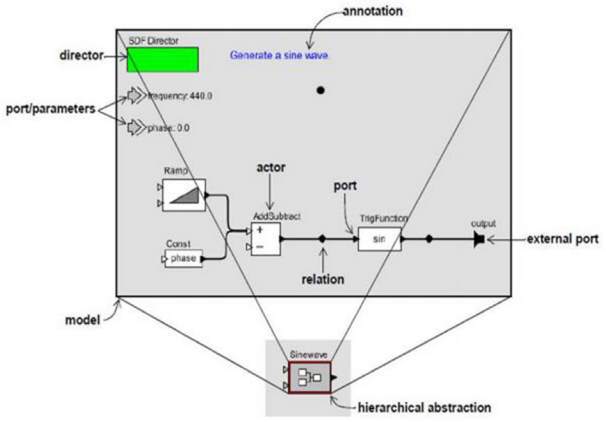

Figure 2. Example of Ptolemy GUI [31]

III. THE POWER ESTIMATION APPROACH

The developed approach is shown in Figure 3. All the elements were developed using C++/SystemC. The elements of the ap-proach are described as following.

A. Data Sender

The Data Sender is responsible for exchanging data among inter-nal and external modules (blue arrows in Figure 3) and also to synchronize internal with external modules. The manner how to define the synchronization and data exchange between the simulated models will be application dependent. In Section 4 is presented a study case describing in details how models were synchronized.

Application data is sent from a simulated model in a feder-ate to other federates through the RTI service[1] *Update Attribute Values*†, executed by the Federate Ambassador. From the re-ceiver side, the Federate Am-

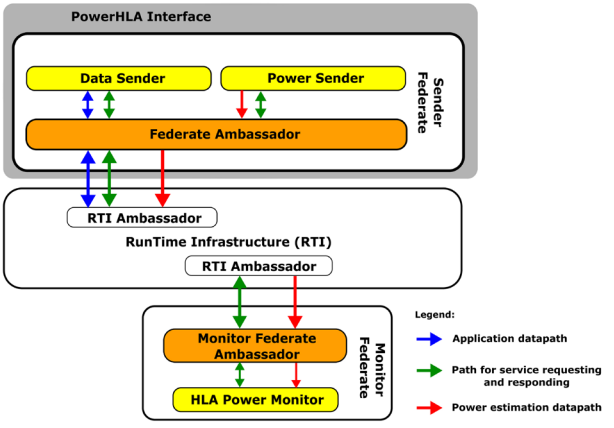

Figure 3. The developed approach

[1]All RTI-initiated services are denoted with a †(dagger) after the service name .

bassador will be notified by the RTI-initiated service (callback) *Reflect Attribute Values*†, which forward the received data to the Data Sender.

B. Power Sender

The Power Sender is responsible for sending the estimated en-ergy accumulated by each monitored component (red arrows in Figure 3). A monitored component is a piece of code, usually a functional module, which the designer wants to monitor. The same Power Sender can monitor various components.

The power consumption estimation is sent to the Monitor Federate according to a predefined time interval, through the Federate Ambassador via Update Attribute Values† service. Thus, all the estimated energy data accumulated from each monitored component is sent to the Monitor Federate, and then all the respective energy related variables are set to zero in order to restart the accumulation of estimated energy.

All power consumption estimation data transmitted from the Sender Federate is received orderly by the Monitor Federate, which is responsible to accumulate the power estimation from all simulators (Federates).

C. Federate Ambassador

The RTI communication with a Sender Federate is established via a Federate Ambassador. Each Sender Federate has a Federate Ambassador instance. This ambassador is responsible for receiv-ing application data from all models, as well as for forwarding responses according to service requests made from RTI, also known as callbacks (green arrows in Figure 3).

For example, when RTI receives a data from other model, the Reflect Attribute Values† callback is executed at Federate Ambas-sador, which will notify the destination module.

Regarding time management, a Sender Federate can request a time advance on RTI. When the request is granted, the RTI sends a response to the Sender Federate via Sender Federate Ambassador, through the callback Time Advance Grant. There other RTI-initiated services, among which can be cited: Discover, Reflect, Time Regulation Enabled, Time Constrained Enabled, etc.

The set composed by Data Sender, Power Sender and Feder-ate Ambassador will be denominated Sender Federate. Every model which uses the PowerHLA approach, have necessarily a Sender Federate.

D. PowerHLA Interface

This element provides all the necessary methods to collect power estimation data during the simulation. These methods are call-ings to the Sender Federate methods.

The methods for data application exchanging and for syn-chronizing must be coded by the PowerHLA user, since these methods are application dependent. Consequently, the user must implement these methods into the PowerHLA interface code.

The PowerHLA Interface is also responsible for creating its respective Sender Federate and to deal with HLA initialization aspects. It was implemented according with the Singleton de-sign pattern. Singleton ensures that a class has only one instance, and provides a global pointer to access it. This design pattern was used in order to provide an easy way to call the PowerHLA Interface methods from any place in the code, without instan-tiating the object every time it is necessary to call any of its methods.

The manner which the Power HLA Interface was imple-mented facilitates its extension to work with other languages and simulators. In our federation, we have a PowerHLA Inter-face instance for each Sender Federate.

E. RunTime Infrastructure

The RTI (RunTime Infrastructure) is the software that provides common interface services, during a High Level Architecture (HLA) federation execution, for synchronizing and data exchang-ing [29]. The RTI contains the implementation of the services initiated by the federates. In this project, we have not imple-mented the RTI, but have used an open-source RTI called CERTI [33].

F. RTI Ambassador

Communication from a federate to the RTI is established via the RTI Ambassador. Each federate requests services on RTI via RTI Ambassador. Services initiated by federates on RTI are denominated Federate Initiated Services (FIS). There are several FIS, among which can be cited: Publish, Subscribe, Register, Update, etc.

The RTI Ambassador is part of the functional structure of the RTI. Therefore, as well as the RTI, the RTI Ambassador was not implemented during the development of the PowerHLA approach, but used the one from CERTI.

G. Monitor Federate

The Monitor Federate Ambassador is the interface between the RTI and the HLA Power Monitor. The Monitor Federate Ambas-sador and the HLA Power Monitor form the Monitor Federate. The federation have a unique instance of the Monitor Federate, which collect all power estimation data from all the federates in the simulation. The Monitor Federate Ambassador is responsi-ble for receiving power estimation

data from all other models, as well as for communicate with RTI via callbacks for control and synchronization.

Since it is a federate, the Monitor Federate also request ser-vices from RTI (e.g., Create Federation Execution, Time Advance Request, etc). It is also responsible for creating the federation and signalize synchronization points to ensure that it will start only when all other Federates signalize they are ready as well.

During the simulation, CSV (Comma Separated Values) files are generated by the Monitor Federate. For each Sender Feder-ate, the Monitor Federate generates a CSV file which contains data about the accumulated estimated energy from its respec-tive monitored components during each predefined collecting interval.

H. Other Relevant Details

The developed approach has a file at Sender and Manager code side, called defines.h, where the designer must define a set of constants in order to configure the simulation environment. The environment comprises two sides, one to deal with power, and other to deal with HLA. Some of most important constants from Sender Federate that deal with power are shown as following:

- POWER_COLLECTING_INTERVAL: The time interval which the Sender Federate must send the power collected data to the Manager Federate. The unit is defined by the INTERVAL_UNIT constant;
- INTERVAL_UNIT: The time interval unit. It can be, for example, SC_US to define the time interval unit as microsec-ond;
- NUMBER_OF_MONITORED_COMPONENTS: The total number of monitored components (it can be a module, a set of modules, a complete SoC, etc.).

We also have constants to deal with the HLA:

- SENDER_FEDERATE_NAME: The Sender Federate name. This name must be unique in the Federation;
- SENDER_FEDERATE_LOOKAHEAD: The federate Looka-head (see [29] for details);
- SENDER_FEDERATE_TIME_STEP: The federate Time Step (see [29] for details).

The Monitor Federate has only the constants needed to deal with the HLA. These constants are similar to the Sender Federate constants.

IV. RESULTS

In this section, a set of experimental results using the proposed approach through one case study is presented. The case study uses two models: an MP-SoC SystemC TLM model built by the MPSoCBench platform and an ESL Ptolemy model. The MP-SoC SystemC TLM model consists of a MIPS processor with one core, memory and a Network-on-chip for interconnection. The Ptolemy model consists of an ESL model for generating messages (character stream), which will be synchronously con-sumed by the MP-SoC model. The present case study aims to demonstrate the PowerHLA capability of creating a distributed and heterogeneous simulation environment.

The experiment demonstrates that models using different languages, as well as different power estimation approaches, could be distributed and simulated using different tools in a synchronized way, besides allowing, in a centralized manner, the grouping and collecting of power estimation data of these models.

A. The Case Study

The MPSoC model executes a cryptographic hash function called Secure Hash Algorithm (SHA). Given an input data (e.g. a string), known as message, the SHA algorithm produces an output, called message digest. Cryptographic hash functions are vastly applied in the information security scope and data integrity checking.

The Ptolemy message generator creates data (bytes), which will be consumed by the SHA application running in the MPSoC model. The data generation is synchronized with the MPSoC SHA application. When the SHA application requests a byte, an HLA function is called, which makes the application in MP-SoC waits for a message coming from Ptolemy model. The Ptolemy model only sends a new data (byte) when it receives a confirma-tion from MPSoC, ensuring the data reception. These process are blocking, i.e., they pause the simulated time. In Figure 4 is shown the simulation environment for the SHA application.

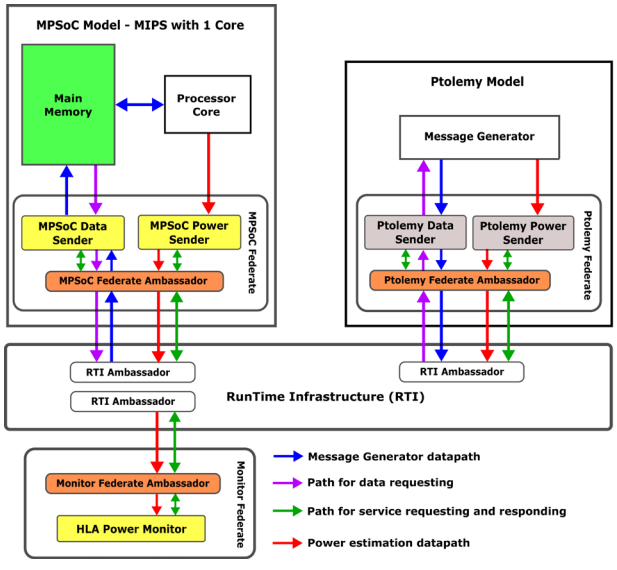

Figure 4. Execution Environment for the SHA Application

The bytes created by the Message Generator are sent through the Ptolemy Federate via RTI, to the MP-SoC Federate. The MP-SoC Federate, in turn, receives these bytes and makes them available for writing them in the MPSoC main memory. This data is read by the SHA application running in MIPS. This ap-plication datapath is represented by the blue arrows in Figure 4.

For each byte received by the MPSoC Federate, the main memory must request a new byte to the Message Generator at Ptolemy model, as SHA needs a new byte to compose the mes-sage. For each data requested, an HLA function is called (via MPSoC Federate), which blocks the SHA execution until the re-quested data arrives. After receiving the data, the main memory requests a new data, confirming the receipt of the previous data. The bytes are requested until the message size is reached. This path is represented in Figure 4 by the purple arrows.

After receiving all bytes which compose the message, the SHA application is executed, and a report is generated in the end, indicating that the SHA finished its execution. As output, the application generates the message digest.

All federates request services to the RTI and also receive responses related to these requests. This path is represented by the green arrows in Figure 4.

In a detailed view, the MPSoc Federate is composed by three elements: The MPSoC Data Sender, the MPSoC Federate Ambas-sador and the MPSoC Power Sender. The MPSoC Data Sender is the element responsible for making available the application data, received from Ptolemy Message Generator, which will be written into the MPSoC main Memory. The power estimation data generated by the processor core (MIPS) is transmitted by the MPSoC Power Sender. The application data and the power estimation data are received and send, respectively, through the MPSoC Federate Ambassador.

Similarly, the Ptolemy Federate is composed by three ele-ments: the Ptolemy Data Sender, the Ptolemy Federate Am-bassador and the Ptolemy Power Sender. The Ptolemy Data Sender is responsible for transmitting the bytes generated by the Ptolemy Message Generator to the MPSoC Federate. The estimated power consumption data from the Message Genera-tor is transmitted by the Ptolemy Power Sender. The Ptolemy Federate Ambassador has an equivalent role to MPSoC Federate Ambassador.

Both federates use the RunTime Infrastructure (RTI) as a means to transmit the application data. The RTI implementation used in the case study was the CERTI [33].

The experiment is comprised in sending eight bytes from the Ptolemy model to the MPSoC model. Each sent byte represents an ASCII character. The decimals 65, 66, 67, 68, 69, 70, 71 and 72 were set, representing the characters "ABCDEFGH".

Once the bytes are received, they are written in the main memory of MPSoC model. Upon receiving all eight bytes, which composes the message, the SHA application reads the characters stream (message) and calculates the message digest.

The collecting of power estimation data from the simulated models is made by the Monitor Federate (Figure 4). This federate is composed by two elements: the Monitor Federate Ambassador and the HLA Power Monitor. The Monitor Federate organizes and groups the received power estimation data from the models, generating as output files in CSV (Comma-Separated Values) format. The power estimation datapath is represented by the red arrows in Figure 4. The Monitor Federate, the RTI and the Ptolemy model were executed in the same machine. The MPSoC model (SystemC) was executed in another computer. The two computers were connected by Ethernet cables via a switch.

The power consumed by the core in the MPSoC model was estimated during the simulation, through the power estimation approach called PowerSC [25]. For the Message Generator, from the Ptolemy model, arbitrary energy values between 0.149 and 0.157 Joules were generated at each sending byte, mimicking a power estimation approach.

The processor core from MPSoC accumulates energy values in Joules for each instruction executed. The energy is accu-mulated during a user predefined interval. Thus, when the predefined interval is reached, the MPSoC Federate sends the accumulated energy data to the Monitor Federate and reset all respective energy values related to the processor core.

During the simulation, 97 samples were obtained. Graphi-cally, the power dissipated by the processor core is presented in Figure 5.

The Message Generator starts to send power estimation data when the first byte is generated, i.e., at the HLA time 13 (Figure 6). For each byte generated (at 1 HLA time unit), the Ptolemy model sends power estimation data. Bytes are created by the Message Generator until the HLA time is 20. Thus, the sending interval for power estimation data from this component com-prises the interval between the HLA time 13 and HLA time 20. The power dissipation data from the Ptolemy Message Genera-tor are shown in Figure 6. The combination of Figures 5 and 6 is presented in Figure 7.

Note that power estimation data are generated by the Pro-cessor Core before the main memory reads the first byte (Figure 7). This power consumption is a consequence of the application loading in the main memory as well as the variables initializa-tion, which are needed during the application execution.

Power estimation data is also generated when SHA reads all the 8 bytes from the main memory. This interval is comprised between the HLA time 22 and

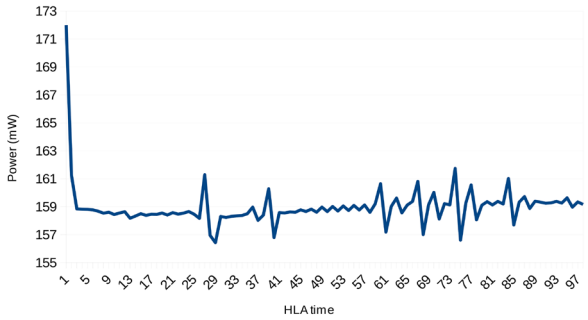

Figure 5. Estimated power consumption of MIPS processor.

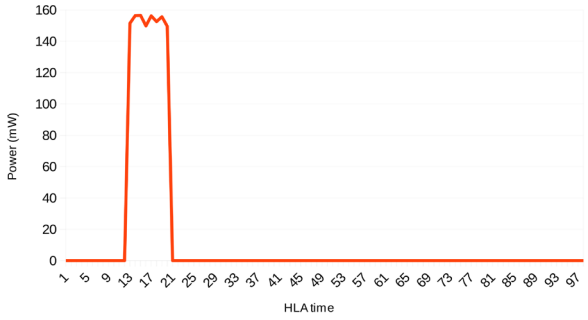

Figure 6. Estimated power consumption of Message Generator processor.

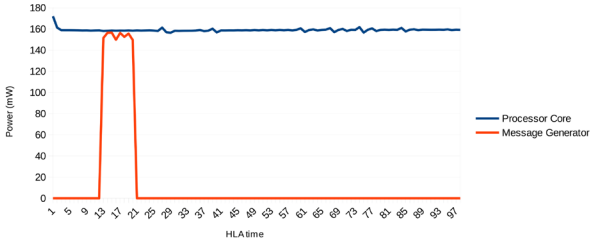

Figure 7. Estimated power consumption of Message Generator and MIPS.

97. This interval consists the application execution phase, i.e., the interval which the message digest is calculated.

B. Models Synchronization

The MPSoC Federate and Ptolemy Federate were defined as time-regulating federate and time-constrained federate. Furthermore, the MPSoC Federate subscribes the object class attributes from Ptolemy Federate and vice versa. These characteristics guarantee that the Ptolemy Federate do not advance the time to a point beyond it could receive Time-Stamped Order (TSO) messages from MPSoC Federate. Similarly, the MPSoC Federate cannot advance in time to a point beyond it could receive TSO messages from Ptolemy Federate.

To synchronize the models, it was necessary to verify the memory cycle from MPSoC model during the SHA application execution. The memory cycle means the time which each byte (character) is read from memory. Although the MPSoC model was not an RTL (Register Transfer Level) model, the reading of each byte from memory occurs approximately in an interval of 13 μs of SystemC simulated time.

The reading of the first byte by the SHA application from the MPSoC memory occurs in approximately 182 μs of SystemC simulated time. Before reading the first byte, the processor is initiated and the SHA application is transferred to the memory. Only after this moment, the application will be ready to execute.

Facing this periodicity regarding the byte reading, it was decided that the HLA time of the models would advance in a discrete and constant manner. During the discrete time simula-tion, the simulation time is divided into a sequence of timesteps with same size. Thus, both federates advance from a timestep to the next. If this periodicity were not detected, the models could be synchronized by events, i.e., the time advance could be performed as all events for a given time were executed.

For example, if the simulation is at HLA time 1, it would advance to HLA time 2 only if all the events from HLA time 1 had been executed. An event could be, for example, the mo-ment which the MPSoC model needed to read a byte in the main memory. Its important to note that the HLA allows both im-plementation modes of synchronization: discrete time and by events.

Thus, each advancing in time performed by the MPSoC Fed-erate occurs at each 13 μs of SystemC simulated time. It was also defined that the memory cycle from MPSoC model would be equivalent to a cycle for data sending from Ptolemy model. The Ptolemy model sends a byte every 1 Ptolemy time unit, fol-lowed by the RTI time advance request. Thus, the 13 μs cycle from SystemC simulator, which executes the MPSoC model, is equivalent to 1 unit time from Ptolemy simulator.

Since the federates from both models are time -regulating and time-constrained, and the MPSoC Federate subscribes class ob-ject attributes published by the Ptolemy Federate and vice versa, when the Ptolemy model sends a data and requests time advanc-ing, it will wait the MPSoC model requests a time advancing, before sending a new byte.

The federates from both models calculate the next time ad-vance value by adding its respective lookahead to the current federate time. Since the time advance is made in a discrete man-ner, the lookahead value from both federates was defined as 1, i.e., every time advance requested by any of these federates, when

granted, will imply the sum of 1 time unit in the current federate time.

It was also defined that the Ptolemy model would only gen-erate data at the moment which the MPSoC model needs to read the first byte. Thus, the Ptolemy model was configured to gen-erate bytes from HLA time 13. The HLA time 13 represents 13 cycles of 13 μs each in SystemC simulated time, i.e., 169 μs. This is the necessary time to initialize the processor and the applica-tion (SHA) on MPSoC. Consequently, at 182 μs from SystemC simulated time, i.e., at HLA time 14, the value of first byte (65 in decimal notation) will be available in the MPSoC main memory.

When the MPSoC receives a byte from Ptolemy model, through the callback Reflect Attribute Values†, implemented into the MPSoc Federate Ambassador, the value is read and writ-ten in the MPSoC main memory. The algorithm responsible for writing the value received by the MPSoC Data Sender into the MPSoC main memory is presented in the Algorithm 1. As mentioned before, the bytes begin to be read from the main memory by the SHA application at 182 μs (SystemC simulated time).

The line 1, from the Algorithm 1, represents the moment which the MPSoC model must advance on time. The next time to advance will always be incremented by 13 μs.

The next time to read (line 2, Algorithm 1) defines the Sys-temC simulated time which the MPSoC model must initiate the reading of bytes from the Ptolemy model. The Ptolemy model was synchronized to send data at HLA time 13. For the MPSoC model, the next time to advance was defined initially as 182 μs. Thus, at HLA time 14, the MPSoC model will start reading the bytes sent by the Ptolemy model. Every byte written in MPSoC memory implies a sum of 13 μs to the time for the next reading (line 6, Algorithm 1).

Thus, at the SystemC time 182 μs, i.e., at the HLA time 14, the MPSoC model will read the first byte sent by the Ptolemy model at the HLA time 13. Next, the MPSoC model will write this byte in the first memory address used by the SHA application to store the message content.

The SHA application will continue requesting new data from Ptolemy model until the message size is reached. It reads a set of bytes written in consecutive address. The address of the first byte read by the application in the MPSoC main memory was discovered by experimentation. This address is denominated base address (line 4, Algorithm 1).

For each received byte, the next address to write the byte is incremented by 1. This increment ensures that the incoming bytes are written in consecutive address into the MPSoC main memory. The synchronization between the MPSoC model and Ptolemy model is presented in Figure 8.

Algorithm 1. Code responsible for writing data into the MP-SoC main memory

```
1:  if The SystemC time is equal or greater than next time to
    advance then
2:      if The SystemC time is equal or greater than next time to
        read and the counter is less than message size then
3:          Read the next byte from Ptolemy model
4:          Set the next address to write the byte as the base ad
            dress plus the counter value
5:          Write the byte into MPSoC memory in the next address
            to write
6:          Set the next time to write as its value plus 13 µs
7:          Increment the counter by 1
8:      end if
9:      Advance the time to a value calculated by summing the
        current federate time plus lookahead value
10:     Set the next time to read as its value plus 13 µs
11:     Set the next time to advance as its value plus 13 µs
12: end if
```

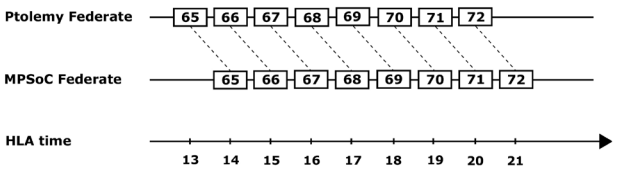

Figure 8. Data synchronization between the MPSoC Federate and Ptolemy Federate

C. Experimental Results

In order to evaluate the impact of simulation on performance when using PowerHLA approach, two experiments were conducted. The first experiment consisted on verifying the simula-tion time of the MP-SoC model when executing 1000 rounds of the SHA application, using messages with 8 bytes each. In this experiment, the MPSoC model exchanged data with the Ptolemy model in a synchronized and distributed way via HLA, without using the PowerHLA set of classes and methods for transmitting and receiving power estimation data. This set of classes are rep-resented by the following elements in Figure 4: MPSoC Power Sender, Ptolemy Power Sender and Monitor Federate.

The second experiment was similar, however, using the Pow-erHLA classes and methods for transmitting and receiving power estimation data. In the Table 1 the simulation time is presented comprising both experiments.

Table 1. Simulation Summary

Experiment	Simulation Time (s)
1. PowerHLA without power collecting	46
2. PowerHLA with power collecting	54

The second experiment, when compared with the first one, has an increasing of 17% in simulation time. Adding the Pow-erHLA set of classes and methods means to add power estima-tion information in federation data flow. These methods define the code portion for transmitting and receiving power estimation data.

Furthermore, this addition implies a new federate in simu-lation: the Monitor Federate. Consequently, this addition also results in an increase of RTI services calls for synchronization.

In order to evaluate the power estimation values, a third experiment was executed. Instead of receiving messages from the Ptolemy model, the 8 bytes were written directly in the memory, which could be read during 1000 rounds of the SHA application. Thus, in contrast to the last two experiments, the third didn't use the PowerHLA approach.

The third experiment was executed in order to compare the total power dissipated by its processor core and the processor core from the cores of the two last experiments, as well as the simulated time.

In all the experiments, the simulated time was 0.485 seconds. Similarly, the total dissipated power by the processor core in all the experiments was 0.158 W, which demonstrates that the approach does not interfere in the simulation consistence and in the power estimation approach adopted by MPSoCBench platform.

V. CONCLUSIONS

In this work, we presented an HLA based approach to enable the creation of a distributed and heterogeneous design environment, composed by different tools and models. These models were described in different languages at system level (ESL). They used different power estimation approaches during the simulation.

The experimental results show the flexibility and effective-ness of PowerHLA. The models employed in case study were described in SystemC and Java. The HLA allowed the Ptolemy framework and SystemC simulator operate in a combined way synchronously. The MPSoC model used a power estimation ap-proach called PowerSC. For the Ptolemy model, arbitrary energy values were generated at each sending byte, mimicking a power estimation approach.

The use of HLA enables the synchronized and distributed simulation of the elements that compose the simulation environ-ment. The approach allows to collect and group power estima-tion data in a centralized manner.

ACKNOWLEDGMENT

The authors would like to thank CAPES (Coordenação de Aper-feiçoamento de Pessoal de Nível Superior) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) to sup-port this project.

REFERENCES

[1] W. Huang, M. Stan, S. Gurumurthi, R. Ribando, and K. Skadron, "Interaction of scaling trends in processor architecture and cooling," in "Semiconductor Thermal Measurement and Management Symposium, 2010. SEMI-THERM 2010. 26th Annual IEEE," (2010), pp. 198–204.

[2] S. Ahuja, D. Mathaikutty, G. Singh, J. Stetzer, S. Shukla, and A. Din-gankar, "Power estimation methodology for a high-level synthesis framework," in "Quality of Electronic Design, 2009. ISQED 2009. Qual-ity Electronic Design," (2009), pp. 541–546.

[3] T. Bouhadiba, M. Moy, and F. Maraninchi, "System-level modeling of energy in tlm for early validation of power and thermal management," in "Proceedings of the Conference on Design, Automation and Test in Europe," (EDA Consortium, San Jose, CA, USA, 2013), DATE '13, pp. 1609–1614.

[4] M. Giammarini, M. Conti, and S. Orcioni, "System-level energy esti-mation with powersim," in "Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on," (2011), pp. 723–726.

[5] W.-T. Hsieh, J.-C. Yeh, S.-C. Lin, H.-C. Liu, and Y.-S. Chen, "System power analysis with dvfs on esl virtual platform," in "SOC Conference (SOCC), 2011 IEEE International," (2011), pp. 93–98.

[6] M. Kuehnle, A. Wagner, A. V. Brito, and J. Becker, "Modeling and implementation of a power estimation methodology for systemc," Inter-national Journal of Reconfigurable Computing (2012).

[7] F. Klein, G. Araujo, R. Azevedo, R. Leao, and L. dos Santos, "An efficient framework for high-level power exploration," in "Circuits and Systems, 2007. MWSCAS 2007. 50th Midwest Symposium on," (2007), pp. 1046–1049.

[8] S.-K. Rethinagiri, O. Palomar, O. Unsal, A. Cristal, R. Ben-Atitallah, and S. Niar, "Pets: Power and energy estimation tool at system-level," in "Quality Electronic Design (ISQED), 2014 15th International Sympo-sium on," (2014), pp. 535–542.

[9] C. Trabelsi, R. Ben Atitallah, S. Meftali, J.-L. Dekeyser, and A. Jemai, "A model-driven approach for hybrid power estimation in embedded systems design," EURASIP Journal on Embedded Systems 2011, 569031 (2011).

[10] M. Caldari, M. Conti, P. Crippa, G. Nuzzo, S. Orcioni, and C. Turchetti, "Instruction based power consumption estimation methodology," in "Electronics, Circuits and Systems, 2002. 9th International Conference on," , vol. 2 (2002), vol. 2, pp. 721–724 vol.2.

[11] M. Kuehnle, A. Wagner, and J. Becker, "A statistical power estimation methodology embedded in a systemc code translator," in "Proceedings of the 24th Symposium on Integrated Circuits and Systems Design," (ACM, New York, NY, USA, 2011), SBCCI '11, pp. 79–84.

[12] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Multi -processor system-on-chip design space exploration based on multi-level modeling techniques," in "Systems, Architectures, Modeling, and Simulation, 2009. SAMOS '09. International Symposium on," (2009), pp. 118–124.

[13] I. Lee, H. Kim, P. Yang, S. Yoo, E.-Y. Chung, K.-M. Choi, J.-T. Kong, and S.-K. Eo, "Powervip: Soc power estimation framework at transaction level," in "Design Automation, 2006. Asia and South Pacific Conference on," (2006), pp. 8 pp.–.

[14] IEEE, "Ieee standard for modeling and simulation - high level archi-tecture (hla) - framework and rules," IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000) pp. 1–38 (2010).

[15] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, Low Power Methodology Manual: For System-on-Chip Design (Springer Publishing Company, Incorporated, 2007).

[16] "Ieee standard for design and verification of low-power integrated cir-cuits," IEEE Std 1801-2013 (Revision of IEEE Std 1801-2009) pp. 1–348 (2013).

[17] G. S. Silveira, A. V. Brito, H. F. de A. Oliveira, and E. U. K. Melcher, "Open systemc simulator with support for power gating design," Int. J. Reconfig. Comput. 2012, 9:9–9:9 (2012).

[18] C. Talarico, J. W. Rozenblit, V. Malhotra, and A. Stritter, "A new frame-work for power estimation of embedded systems," Computer 38, 71–78 (2005).

[19] N. Bansal, K. Lahiri, A. Raghunathan, and S. T. Chakradhar, "Power monitors: A framework for system-level power estimation using heterogeneous power models," in "VLSI Design, 2005. 18th International Conference on," (IEEE, 2005), pp. 579–585.

[20] Y.-H. Park, S. Pasricha, F. J. Kurdahi, and N. Dutt, "System level power estimation methodology with h. 264 decoder prediction ip case study," in "Computer Design, 2007. ICCD 2007. 25th International Conference on," (IEEE, 2007), pp. 601–608.

[21] W.-T. Hsieh, J.-C. Yeh, and S.-Y. Huang, "Pac duo system power estimation at esl," in "Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific," (IEEE, 2010), pp. 815–820.

[22] T.-J. Lin, C.-N. Liu, S.-Y. Tseng, Y.-H. Chu, and A.-Y. Wu, "Overview of itri pac project-from vliw dsp processor to multicore computing platform," in "VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on," (IEEE, 2008), pp. 188–191.

[23] M. Giammarini, M. Conti, and S. Orcioni, "System-level energy estima-tion with powersim," in "2011 18th IEEE International Conference on Electronics, Circuits, and Systems," (2011).

[24] R. Damaševicius,ˇ "Estimation of design characteristics at rtl modeling level using systemc," Information Technology And Control 35 (2006).

[25] F. Klein, G. Araujo, R. Azevedo, R. Leao, and L. Santos, "An Efficient Framework for High-Level Power Exploration," in "Proceedings of the 50th Midwest Symposium on Circuits and Systems - MWSCAS 2007," (2007), pp. 1046–1049.

[26] D. Greaves and M. Yasin, "Tlm power3: Power estimation methodology for systemc tlm 2.0," in "Models, Methods, and Tools for Complex Chip Design," (Springer, 2014), pp. 53–68.

[27] M. Moy, "Tlm-power tlm-power++ application conclusion mini power-aware tlm-platform," (2010).

[28] G. B. Vece and M. Conti, "Power estimation in embedded systems within a systemc-based design context: the pktool environment," in "Intelligent solutions in Embedded Systems, 2009 Seventh Workshop on," (IEEE, 2009), pp. 179–184.

[29] "IEEE Standard for Modeling and Simulation (M&S) High Level Archi-tecture (HLA)," IEEE Std 1516.x-2010 (2010).

[30] L. Duenha, M. Guedes, H. Almeida, M. Boy, and R. Azevedo, "Mp-socbench: A toolset for mpsoc system level evaluation," in "Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on," (IEEE, 2014), pp. 164–171.

[31] C. Ptolemaeus, ed., System Design, Modeling, and Simulation using Ptolemy II (Ptolemy.org, 2014).

[32] A. V. Brito, A. V. Negreiros, C. Roth, O. Sander, and J. Becker, "Devel-opment and evaluation of distributed simulation of embedded systems using ptolemy and hla," in "Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications," (IEEE Computer Society, Washington, DC, USA, 2013), DS-RT '13, pp. 189–196.

[33] E. Noulard, J.-Y. Rousselot, and P. Siron, "Certi, an open source rti, why and how," in "Spring Simulation Interoperability Workshop," (2009), pp. 23–27.